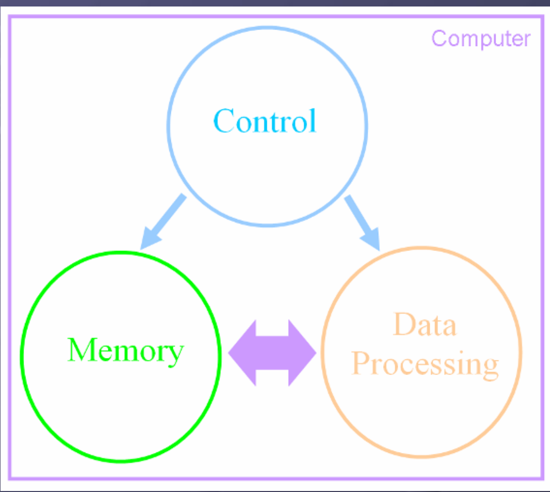
Systems and Devices 1
Lec 5a : The Computer

Before we get started ...

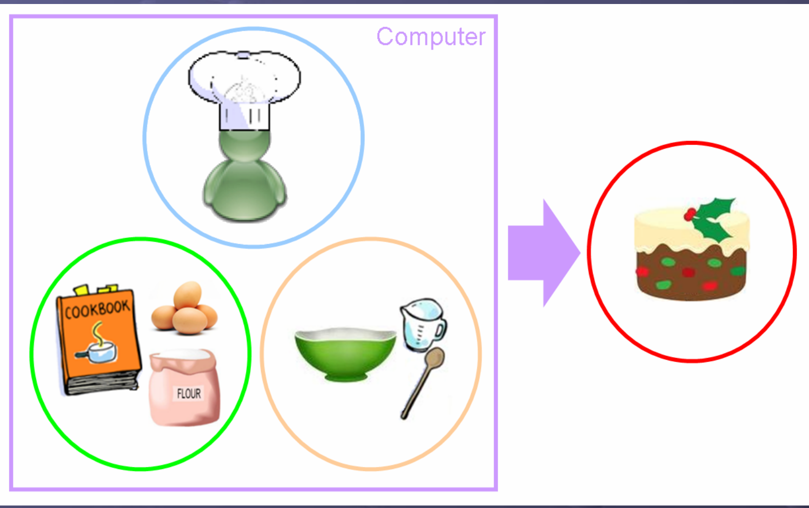
- Logic + Memory = Computer
  - We have all the hardware elements needed to make a computer i.e. hardware DNA.
- BUT, how do we take these components and configure them to execute a program?
  - That final spark of life.
- Need to consider how we will represent a program in memory, how the instructions used are encoded and how these will be processed by the hardware.

Simplified Computer Architecture

- Program : the sequence of instructions stored in memory required to solve a specified problem
  - Hardware architecture required to process instructions
    - Load / store data
    - Perform arithmetic functions ...



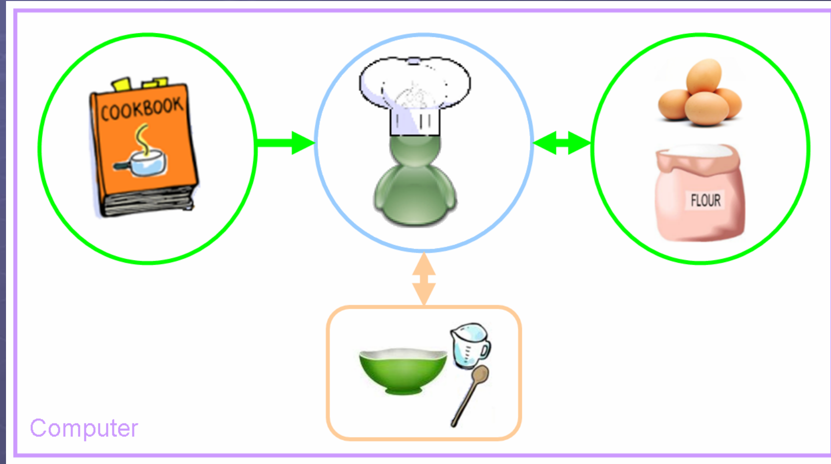
Computer Analogy



- Baking a cake

Slide 5

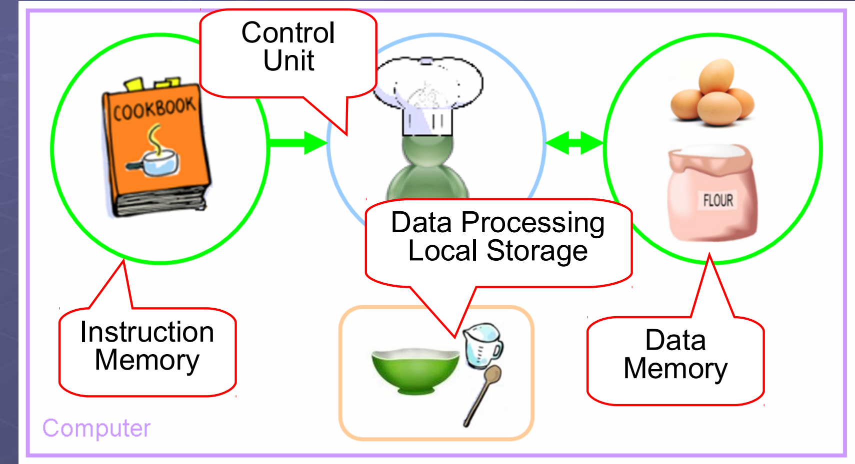
Computer Analogy



- Computer architecture
  - 2 memories, 1 control unit and 1 data processing unit

Slide 6

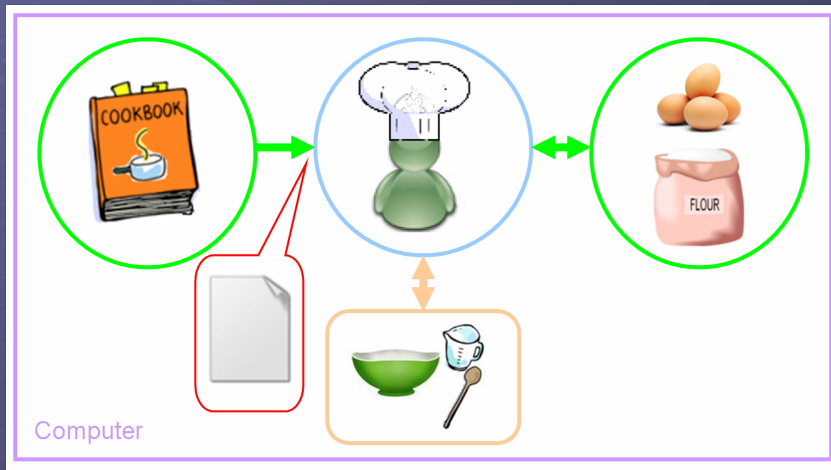
Computer Analogy



- Computer architecture
  - 2 memories, 1 control unit and 1 data processing unit

Slide 7

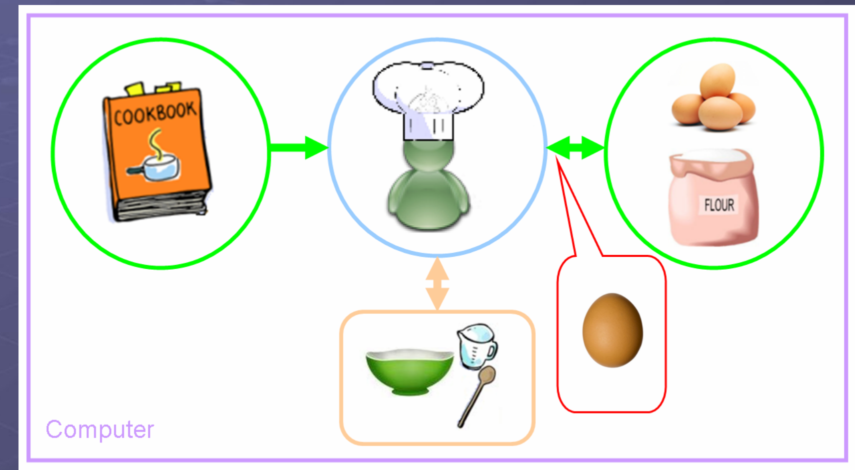
Computer Analogy



- Computer architecture
  - Fetch : read instruction from cook book.

Slide 8

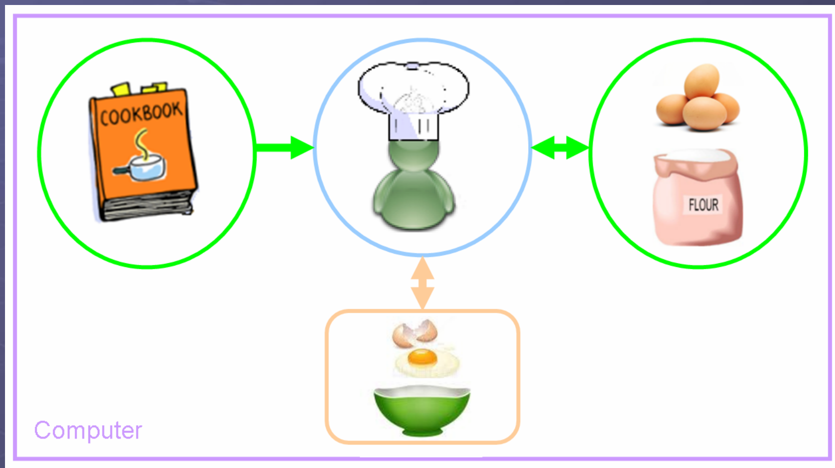
Computer Analogy



- Computer architecture
  - Decode : understand instruction and get ingredients from store

Slide 9

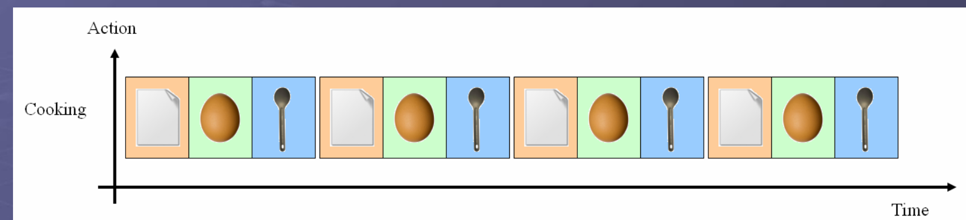
Computer Analogy



- Computer architecture
  - Execute : crack egg into bowl.

Slide 10

Computer Analogy



- Instruction cycle timing diagram
  - Fetch – Decode – Execute
    - Each instruction requires multiple phases to be processed
    - Data and functions encoded, represented by symbols that the hardware within the computer can understand
  - Algorithm : a list of simple instructions, defining the step by step procedure for solving a problem
  - Program : a list of instructions that a computer follows to perform a specified task. May require one or more different algorithms.

Slide 11

What is an instruction?

$$10 \times 5 = 50$$

- A computer is not hard-wired (fixed), but can be controlled by a sequence of selected instructions.
  - Instruction : defines a function, input data and where the result should be stored
    - Made up of operands and an opcode

Slide 12

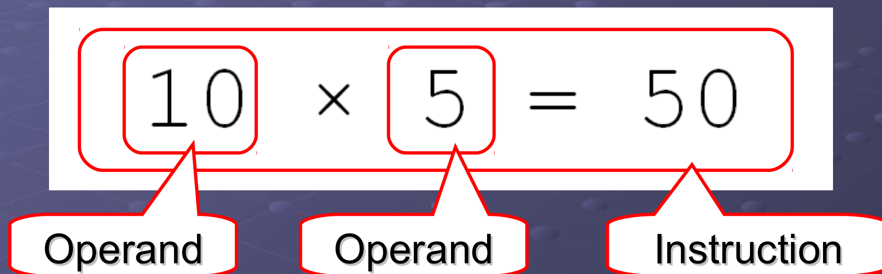
What is an instruction?

$$10 \times 5 = 50$$

Instruction

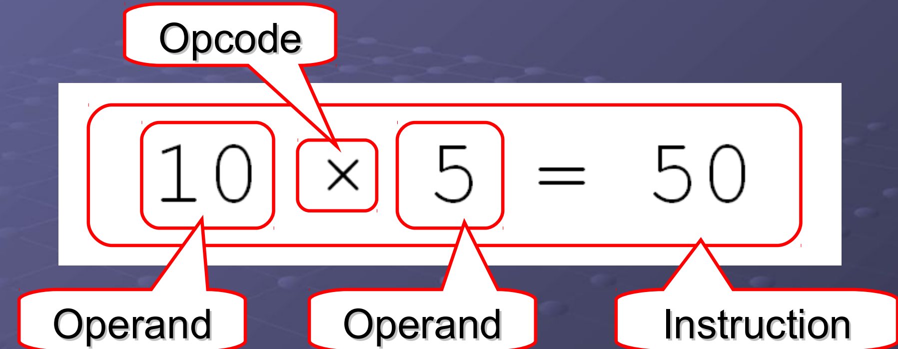
- A computer is not hard-wired (fixed), but can be controlled by a sequence of selected instructions.
  - Instruction : defines a function, input data and where the result should be stored
    - Made up of operands and an opcode

What is an instruction?



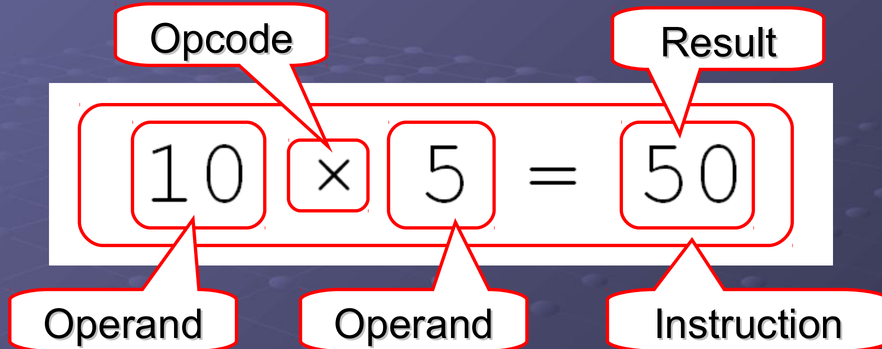
- A computer is not hard-wired (fixed), but can be controlled by a sequence of selected instructions.
  - Instruction : defines a function, input data and where the result should be stored
    - Made up of operands and an opcode

What is an instruction?



- A computer is not hard-wired (fixed), but can be controlled by a sequence of selected instructions.
  - Instruction : defines a function, input data and where the result should be stored
    - Made up of operands and an opcode

What is an instruction?



- A computer is not hard-wired (fixed), but can be controlled by a sequence of selected instructions.
  - Instruction : defines a function, input data and where the result should be stored
    - Made up of operands and an opcode

What instructions do we need?

- Three main instruction groups
  - Move : read and write data to / from storage locations, registers (memory) ...
    - Load – Store : move processed information around the machine.
  - Arithmetic : +, –, ×, ÷
    - Logic : AND, OR, XOR, NOT ...
  - Control
    - Change the sequential flow of execution of operations within the machine.
- Addressing Modes
  - How operands are accessed and result stored
    - Implicit : implied by the opcode / architecture.
    - Immediate / literal : constant, value in instruction.
    - Absolute / Direct : address in memory, value in memory.

How do we represent instructions?

- Programming language classification
  - Machine code
  - Assembler code
  - High level code

How do we represent instructions?

- Programming language classification
  - **Machine code**
    - Binary or hexadecimal representations
      - 0001 0000 0001 0001 (SimpleCPU add 17 to ACC)
    - In general specific to a particular processor
    - “Machine language, a pattern of bits, encoding machine operations”
    - “The sequence of binary patterns that is executed by the hardware; the set of instructions that a computer’s CPU can understand and obey directly without any translation”
  - Assembler code
  - High level code

How do we represent instructions?

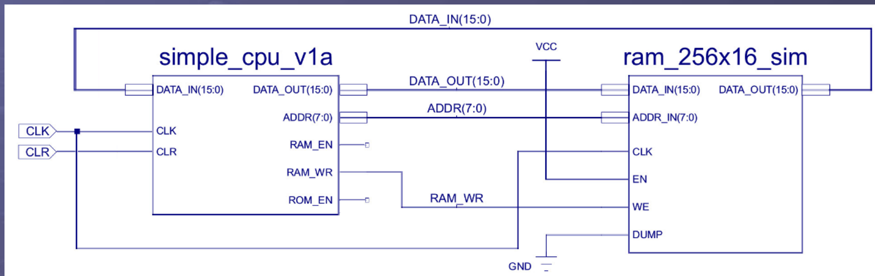
- Programming language classification
  - Machine code
  - **Assembler code**
    - Textual representations
      - ADD 0x11 (SimpleCPU add 17 to ACC)
    - “A programming language that utilises symbols to represent operation codes and storage locations”
    - “Human readable notation for the machine language that a specific computer architecture uses, replacing raw binary patterns with symbols called mnemonics”
  - High level code

How do we represent instructions?

- Programming language classification
  - Machine code
  - Assembler code
  - **High level code**
    - Textual description
      - IF (A=B) THEN C = C + 1
    - “A programming language where each instruction corresponds to several machine code instructions”
    - “A high level programming language is more user friendly, to some extent platform independent, providing a layer of abstraction between the programmer and the low level hardware”

Slide 21

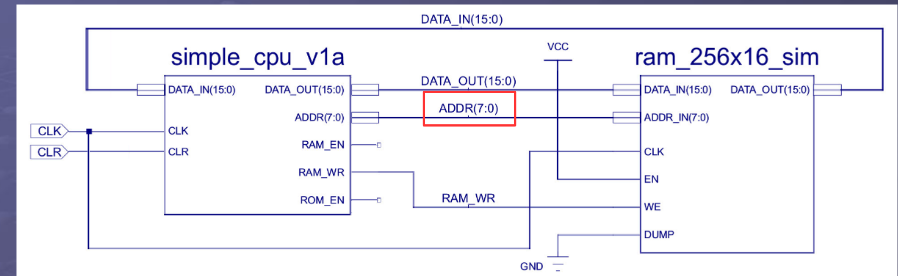
How are instructions stored?



- Von Neumann architecture : stored program computer
  - Instructions and data stored in the same memory
- The processor is connected to memory using 3 buses:
  - Address Bus : ADDR(7:0)
  - Data Bus : DATA_IN(15:0), DATA_OUT(15:0)
  - Control Bus : RAM_EN, RAM_WR, ROM_EN

Slide 22

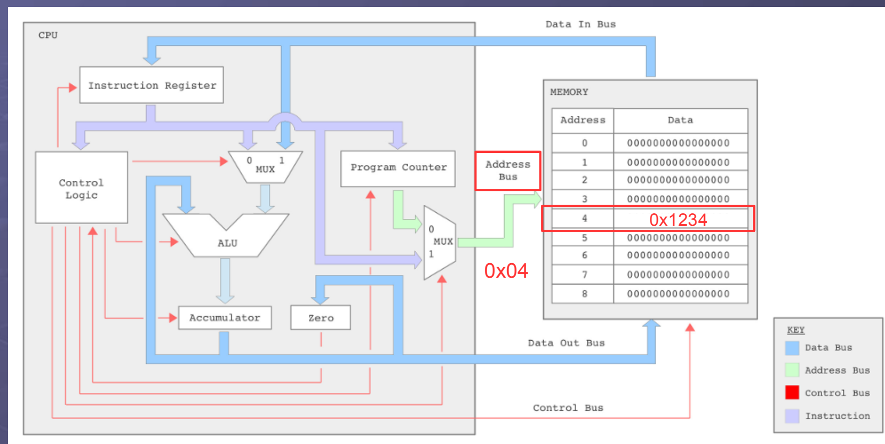
SimpleCPU_v1a



- Address Bus : ADDR(7:0)
  - Identify the particular memory location, device or component that will be involved in a data transfer. The width ***n*** determines the number of locations (things) that can be identified i.e. 2^n, if n=8 then 256 memory locations can be identified, addresses 0 – 255

Slide 23

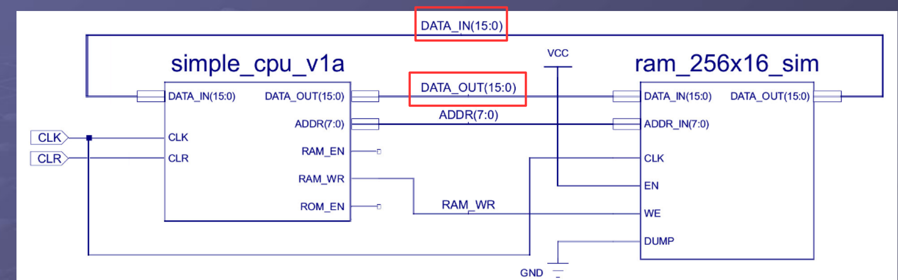
SimpleCPU_v1a



- Address bus : 8bits, 256 locations, source IR or PC.

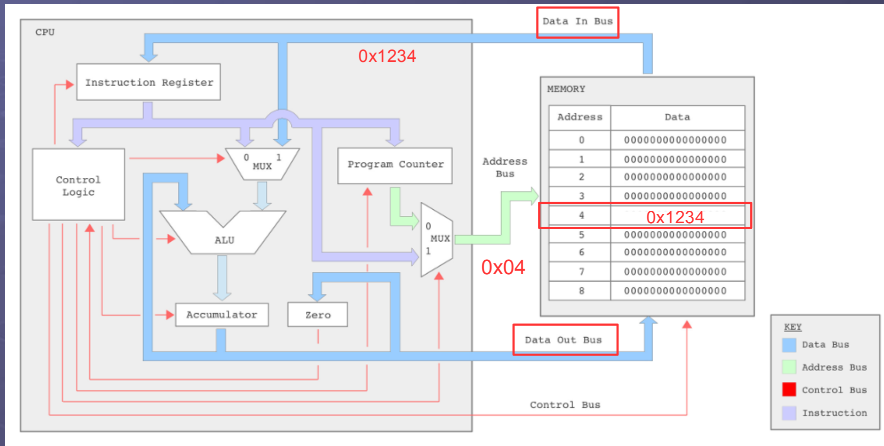
Slide 24

SimpleCPU_v1a



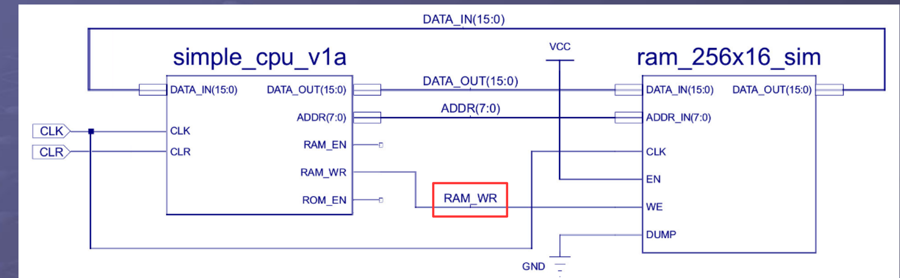
- Data Bus : DATA_IN(15:0), DATA_OUT(15:0)
  - Information (data) that is to be transferred. The width ***n*** determines how fast data is transferred i.e. bits per second, typically matched to internal register width.
  - However, smaller (to reduce IO pins) and larger (increase data bandwidth) widths are also common, architecture dependent.

SimpleCPU_v1a



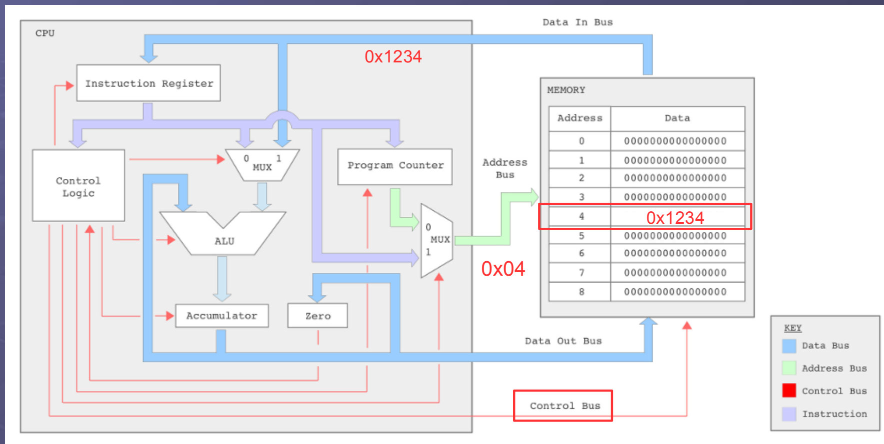
- Data-In bus : 16bits, destination IR or ALU
- Data-out bus : 16bits, source ACC

SimpleCPU_v1a



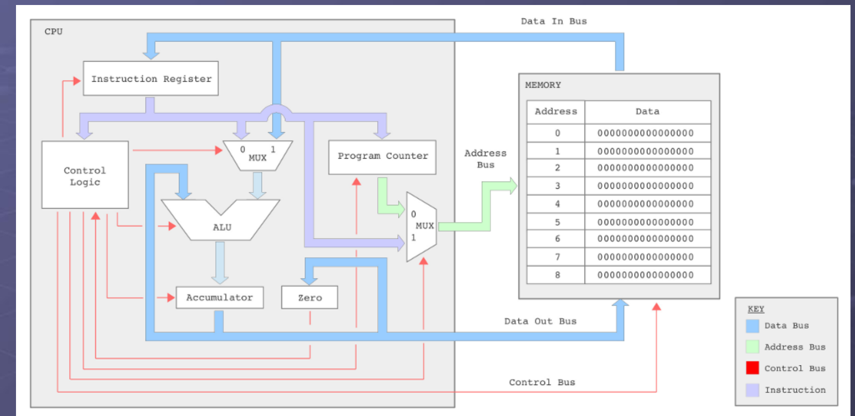
- Control Bus : ROM_EN, RAM_EN, RAM_WR
  - Synchronises the flow of information across the address and data buses. As a minimum the bus must inform the memory devices if the transaction is a read or a write operation.
  - In a multi-master (multi-processor) system arbitration is required to determine who has access to these buses and at what times.

SimpleCPU_v1a



- Control bus : 3bits, enable memory devices, control memory transaction i.e. read or write.

SimpleCPU_v1a



- We need to define the processor's instruction set
  - The list of assembly language instructions supported by a processor (that can be performed by its architecture).

Slide 29

Instruction set

RTL	ENCODING	ASSEMBLER
Move KK : ACC <- KK	0000 XXXX KKKKKKKK	MOVE 0x01
Add KK : ACC <- ACC + KK	0001 XXXX KKKKKKKK	ADD 0x23
Sub KK : ACC <- ACC - KK	0010 XXXX KKKKKKKK	SUB 0x45
And KK : ACC <- ACC & KK	0011 XXXX KKKKKKKK	AND 0x67
Load AA : ACC <- M[AA]	0100 XXXX AAAAAAAA	LOAD 0x89
Store AA : M[AA] <- ACC	0101 XXXX AAAAAAAA	STORE 0x89
AddM AA : ACC <- ACC + M[AA]	0110 XXXX AAAAAAAA	ADDM 0xAB
SubM AA : ACC <- ACC - M[AA]	0111 XXXX AAAAAAAA	SUBM 0xAB
JumpU AA : PC <- AA	1000 XXXX AAAAAAAA	JUMPU 0xCD
JumpZ AA : IF Z=1 PC <- AA ELSE PC <- PC + 1	1001 XXXX AAAAAAAA	JUMPZ 0xEF
JumpNZ AA : IF Z=0 PC <- AA ELSE PC <- PC + 1	1010 XXXX AAAAAAAA	JUMPNZ 0xF0

- Register Transfer Level (RTL) syntax, read “<-” as “updated with”
- KK=Constant, AA=Address, M[ ]=Memory , Z=Zero Flag

Slide 30

Type 00 : Immediate

	Opcode				Not used				Operand							
MOVE 0x12	0	0	0	0	0	0	0	0	K	K	K	K	K	K	K	K
ADD 0x34	0	0	0	1	0	0	0	0	K	K	K	K	K	K	K	K
SUB 0x56	0	0	1	0	0	0	0	0	K	K	K	K	K	K	K	K
AND 0x78	0	0	1	1	0	0	0	0	K	K	K	K	K	K	K	K

- MOVE 0x12 : set ACC to value 0x12 ACC ← 0x12
- ADD 0x34 : add the value 0x34 to ACC ACC ← ACC + 0x34
- SUB 0x56 : subtract value 0x56 from ACC ACC ← ACC – 0x56
- AND 0x78 : bitwise AND ACC and value 0x78 ACC ← ACC & 0x78

Slide 31

Type 01 : Absolute

	Opcode				Not used				Operand							
LOAD 0x9A	0	1	0	0	0	0	0	0	A	A	A	A	A	A	A	A
STORE 0x9A	0	1	0	1	0	0	0	0	A	A	A	A	A	A	A	A
ADDM 0xBC	0	1	1	0	0	0	0	0	A	A	A	A	A	A	A	A
SUBM 0xCD	0	1	1	1	0	0	0	0	A	A	A	A	A	A	A	A

- LOAD 0x9A : read memory address 0x9A store data in ACC
  - ACC ← M[0x9A]
- STORE 0x9A : write ACC data to memory address 0x9A
  - M[0x9A] ← ACC
- ADDM 0xBC : add data stored in memory address 0xBC to ACC
  - ACC ← ACC + M[0xBC]
- SUBM 0xCD : sub data stored in memory address 0xCD from ACC
  - ACC ← ACC - M[0xCD]

Slide 32

Type 10 : Direct

	Opcode				Not used				Operand							
JUMPU 0xDE	1	0	0	0	0	0	0	0	A	A	A	A	A	A	A	A
JUMPZ 0xF0	1	0	0	1	0	0	0	0	A	A	A	A	A	A	A	A
JUMPNZ 0xF0	1	0	1	0	0	0	0	0	A	A	A	A	A	A	A	A

- JUMPU 0xDE : update PC with address 0xDE
  PC ← 0xDE
- JUMPZ 0xF0 : if ACC=0 update PC with address 0xF0 else inc PC
  if Z PC ← 0xF0 else PC=PC+1
- JUMPNZ 0xF0 : if ACC!=0 update PC with address 0xF0 else inc PC
  if NZ PC ← 0xF0 else PC=PC+1

Slide 33

Machine level instructions

```
0000 0000 0000 0000 =   0x0000
0100 0000 1010 1010 =   0x40AA
0000 1111 1011 1011 =   0x0FBB
1000 0000 1100 1100 =   0x80CC
1111 1111 1111 1111 =   0xFFFF
```

- Quick Quizz
  - What instructions do these binary patterns represent?
    - Two trick questions :)

Slide 34

Worked Example : your first program

- Multiply 10 by 3
  - Multiplication by repeated addition
- Don't worry your second program is “Hello World”

```
START:
  Total = 0
  Count = 3
WHILE Count!=0:
  Total=Total+10
  Count=Count-1
END LOOP
```

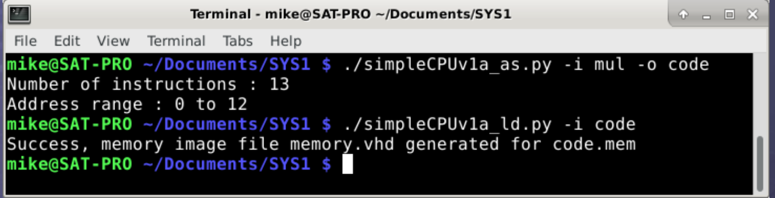
```
START:
    MOVE 0x00
    STORE 0x0D
    MOVE 0x03
    STORE 0x0E
LOOP:
    JUMPZ 0x0C
    SUB 0x01
    STORE 0x0E
    LOAD 0x0D
    ADD 0x0A
    STORE 0x0D
    LOAD 0x0E
    JUMPU 0x04
END:
    STOP
```

- What is stored in memory locations :
  - 0x0D and 0x0E
  - 0x00 to 0x0C
- What are the operand values :
  - 0x00, 0x0D, 0x03, 0x0E, 0x0C ...

Slide 35

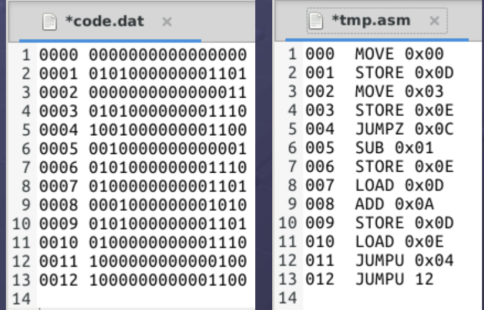
Assembler / Linker

```
START:
    MOVE 0x00
    STORE 0x0D
    MOVE 0x03
    STORE 0x0E
LOOP:
    JUMPZ 0x0C
    SUB 0x01
    STORE 0x0E
    LOAD 0x0D
    ADD 0x0A
    STORE 0x0D
    LOAD 0x0E
    JUMPU 0x04
END:
    STOP
```


```
Terminal - mike@SAT-PRO ~/Documents/SYS1
File Edit View Terminal Tabs Help
mike@SAT-PRO ~/Documents/SYS1 $ ./simpleCPUv1a_as.py -i mul -o code
Number of instructions : 13
Address range : 0 to 12
mike@SAT-PRO ~/Documents/SYS1 $ ./simpleCPUv1a_ld.py -i code
Success, memory image file memory.vhd generated for code.mem
mike@SAT-PRO ~/Documents/SYS1 $
```

- Convert assembly language into machine code to load into memory
- Python program

simpleCPUv1a_as.py
simpleCPUv1a_ld.py

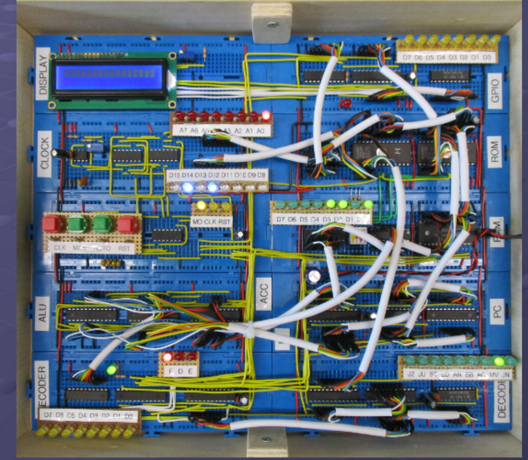

```
*code.dat
1 0000 0000000000000000
2 0001 0101000000001101
3 0002 0000000000000011
4 0003 0101000000001110
5 0004 1001000000001100
6 0005 0010000000000001
7 0006 0101000000001110
8 0007 0100000000001101
9 0008 0001000000001010
10 0009 0101000000001101
11 0010 0100000000001110
12 0011 1000000000000100
13 0012 1000000000001100
14
```

```
*tmp.asm
1 000 MOVE 0x00
2 001 STORE 0x0D
3 002 MOVE 0x03
4 003 STORE 0x0E
5 004 JUMPZ 0x0C
6 005 SUB 0x01
7 006 STORE 0x0E
8 007 LOAD 0x0D
9 008 ADD 0x0A
10 009 STORE 0x0D
11 010 LOAD 0x0E
12 011 JUMPU 0x04
13 012 JUMPU 12
14
```


Slide 36

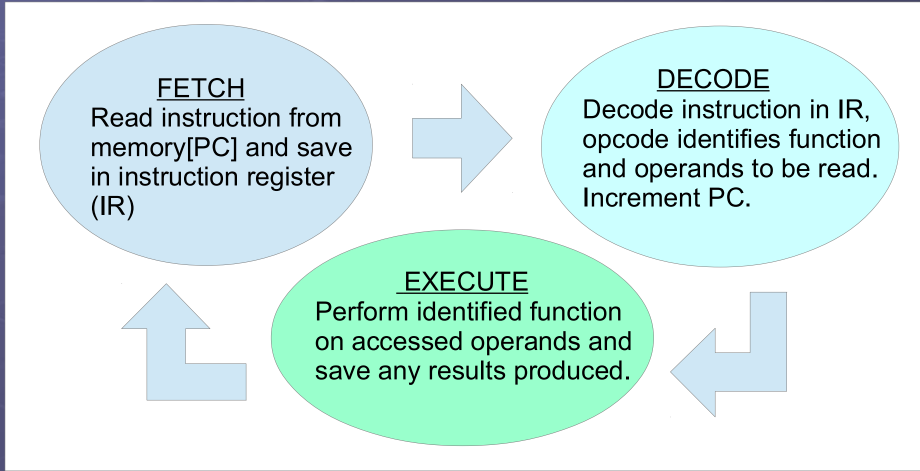
Demo : SimpleCPU_v1a

```
START:
    MOVE 0x00
    STORE 0x0D
    MOVE 0x03
    STORE 0x0E
LOOP:
    JUMPZ 0x0C
    SUB 0x01
    STORE 0x0E
    LOAD 0x0D
    ADD 0x0A
    STORE 0x0D
    LOAD 0x0E
    JUMPU 0x04
END:
    STOP
```



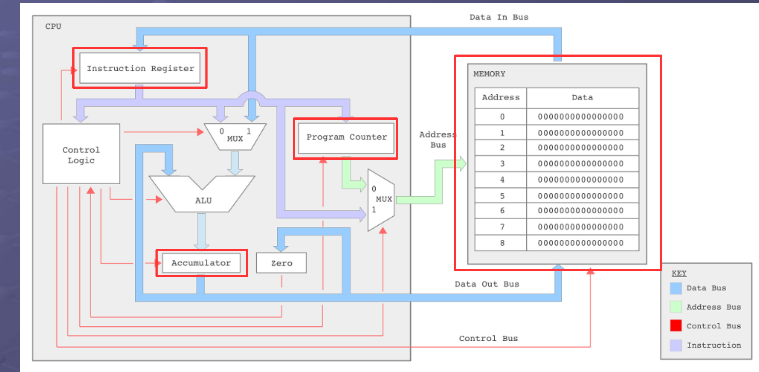
- Using LED displays identify each instruction and its phases.

Fetch-Decode-Execute



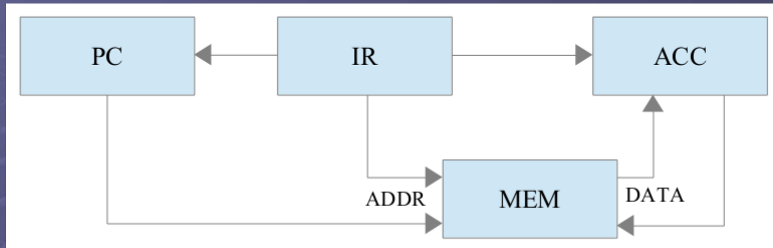
- FDE operations

Whats happening in the CPU?



- Need to implement “house keeping” functions in hardware for Fetch – Decode – Execute cycle.
- Control when memory elements within the processor are updated e.g. IR, PC, ACC and RAM.

RTL

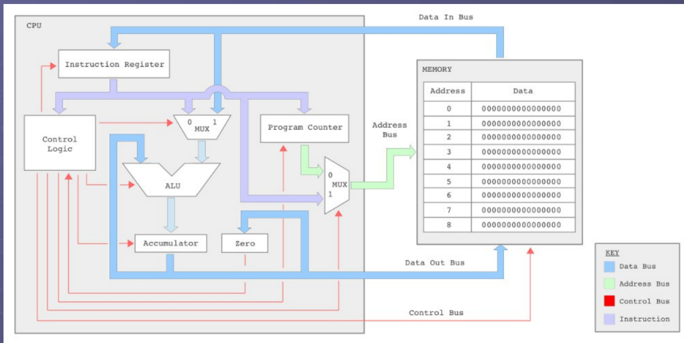


- The operation of a computer can be described in terms of a collection of memory elements and the movement of data between them.
  - Within the CPU each instruction is broken down into a series of steps i.e. micro-instructions.
  - Micro-instructions are typically represented using register transfer level (RTL) descriptions.

Whats happening in the CPU?

- CPUSim : step through machine instruction phases
  - Micro-instructions

RTL

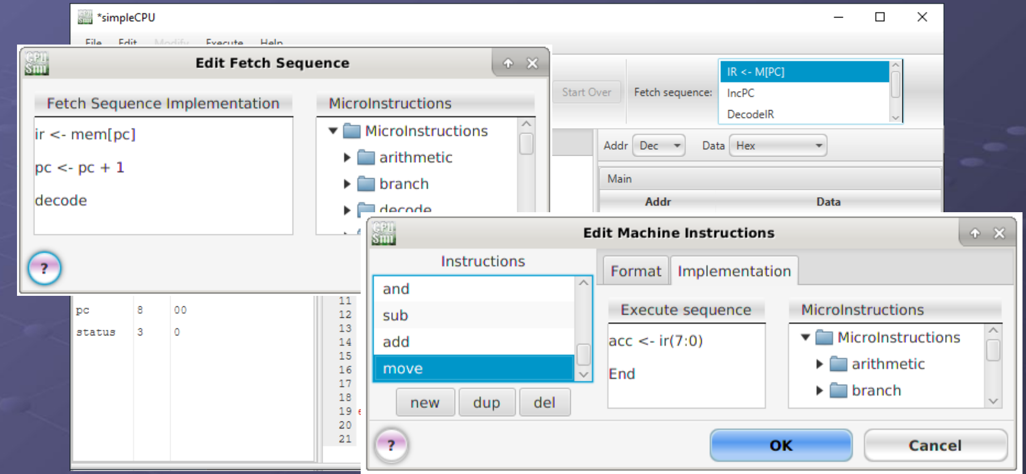


EXAMPLES

```
ACC <- ACC + 1
ACC <- IR(7:0)
ACC <- M[PC]
M[IR(7:0)] <- ACC
```

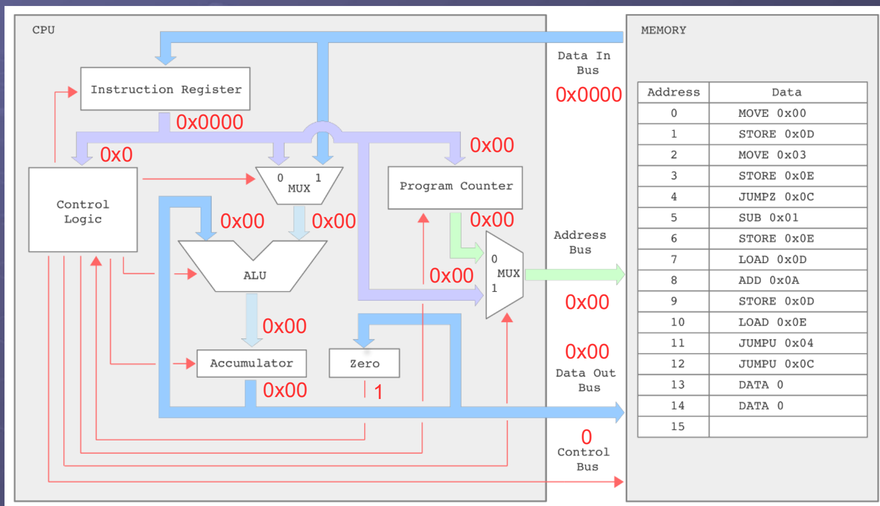
- Quick Quizzz
  - Write out the micro-instructions in RTL format for the FETCH cycle and DECODE / EXECUTE phases of the MOVE instruction.

Demo : CPUSim



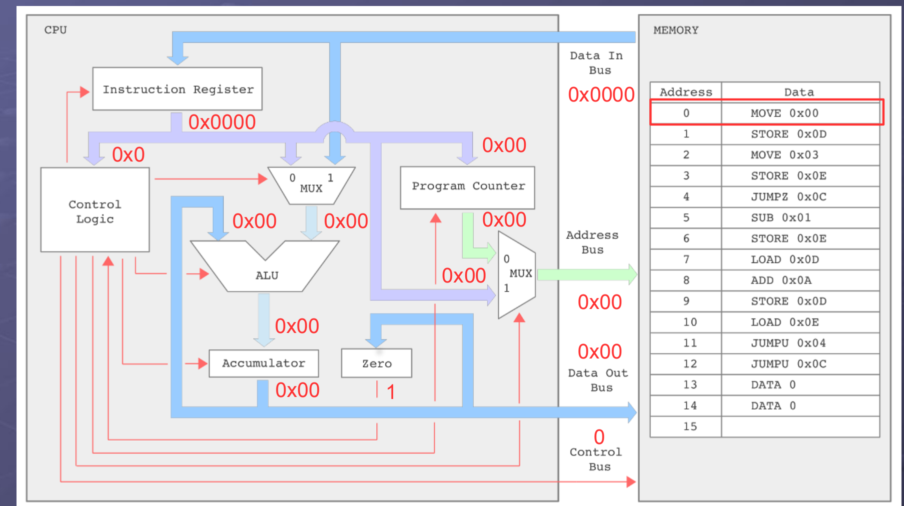
- Each machine level instruction is implemented by multiple micro-instructions

SimpleCPU_v1a : MOVE



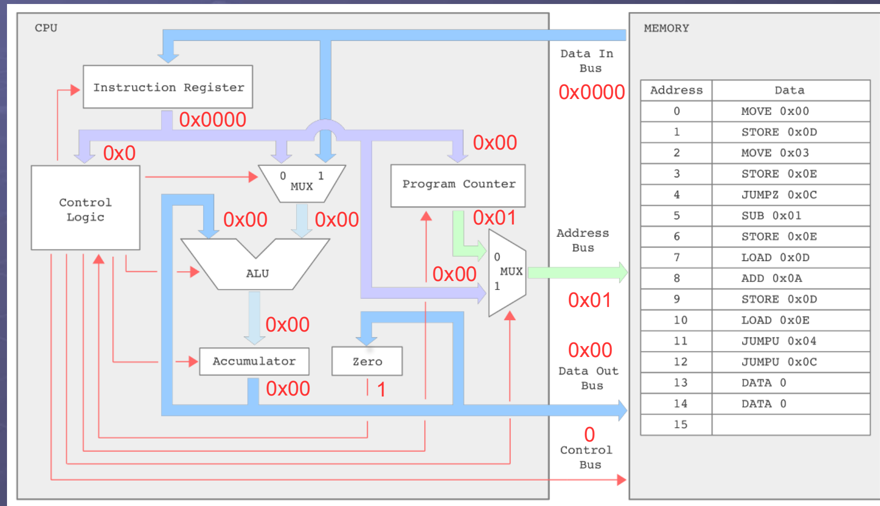
- Reset : pulse clear line, reset all DFF to 0

SimpleCPU_v1a : MOVE



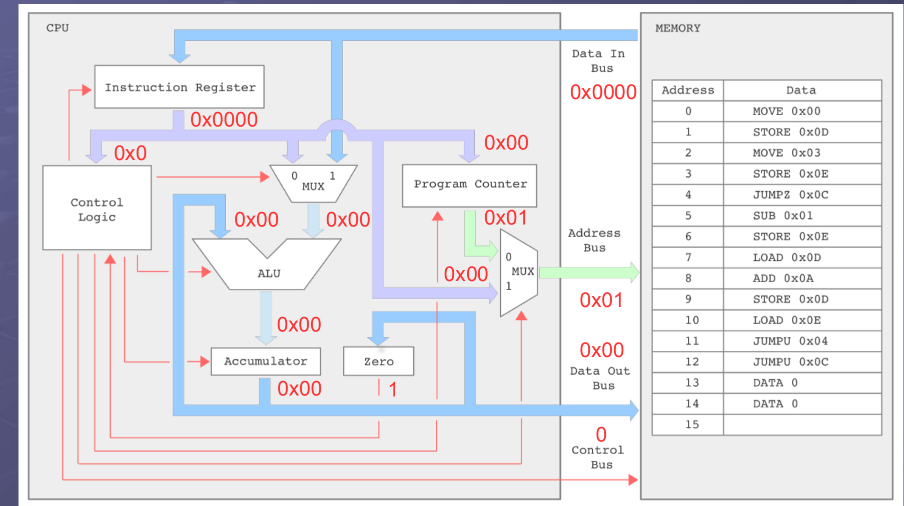
- Fetch : get instruction stored at PC address

SimpleCPU_v1a : MOVE



- Decode : inc PC, set MUX and ALU control lines

SimpleCPU_v1a : MOVE



- Execute : update ACC

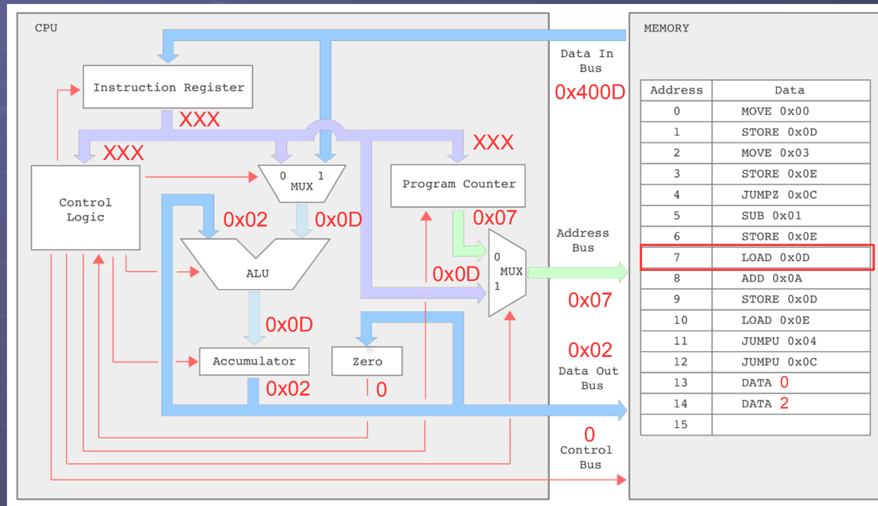
What if …

- Quick Quizzz
  - Bad example :), nothing really changed in the processor, lets consider what will happen if:
    - MOVE 0x03 instruction is executed
  - What will happen if the processor tries to run a program before it is loaded into memory i.e. all memory locations contain the value 0x0000?

Instruction Set Simulator

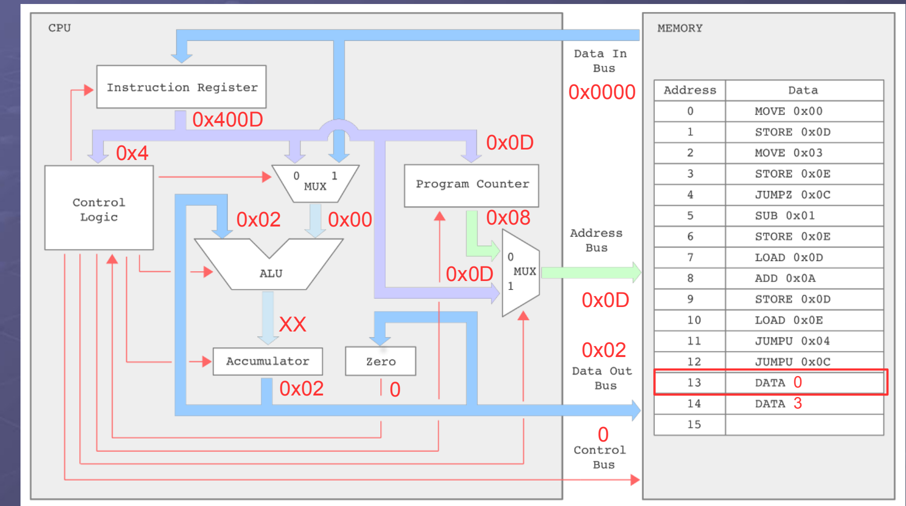
- LOAD: type 01 instruction, absolute addressing mode
  - Operands : ACC and IR(7:0)

SimpleCPU_v1a : LOAD



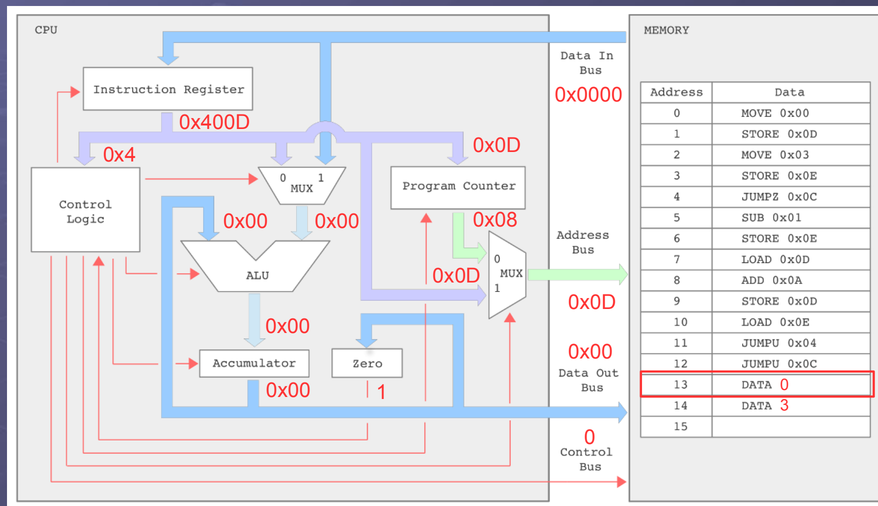
- Fetch : get instruction stored at PC address

SimpleCPU_v1a : LOAD



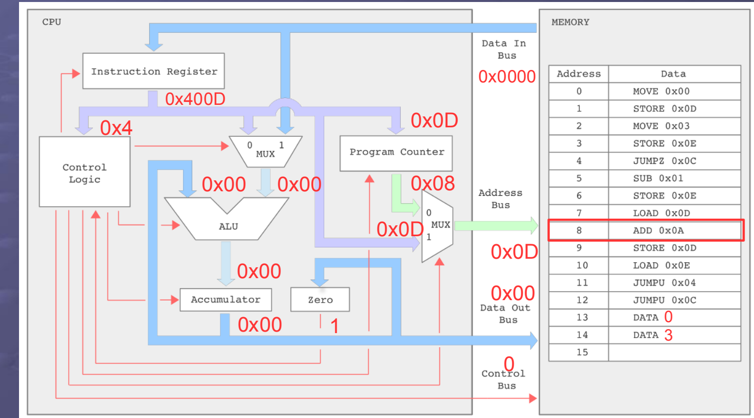
- Decode : inc PC, set MUX and ALU control lines

SimpleCPU_v1a : LOAD



- Execute : perform subtraction and update ACC

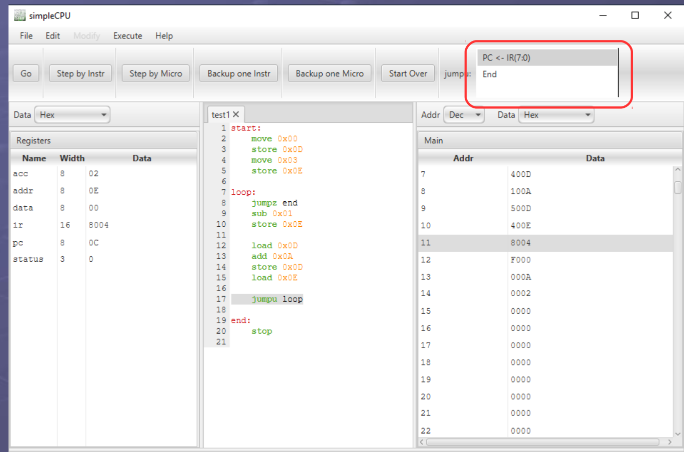
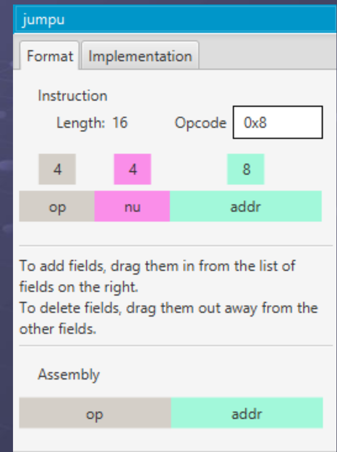
SimpleCPU_v1a : ADD



- Quick Quizzz
  - Can you step through the micro-instructions for the next instruction : ADD 0x0A?

Slide 53

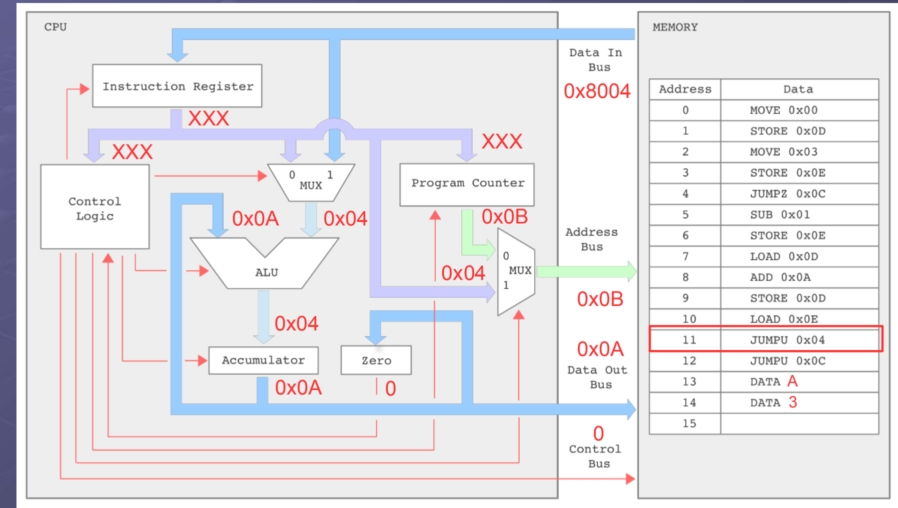
Instruction Set Simulator

- JUMPU: type 10 instruction, direct addressing mode
  - Operands : IR(7:0)

Slide 54

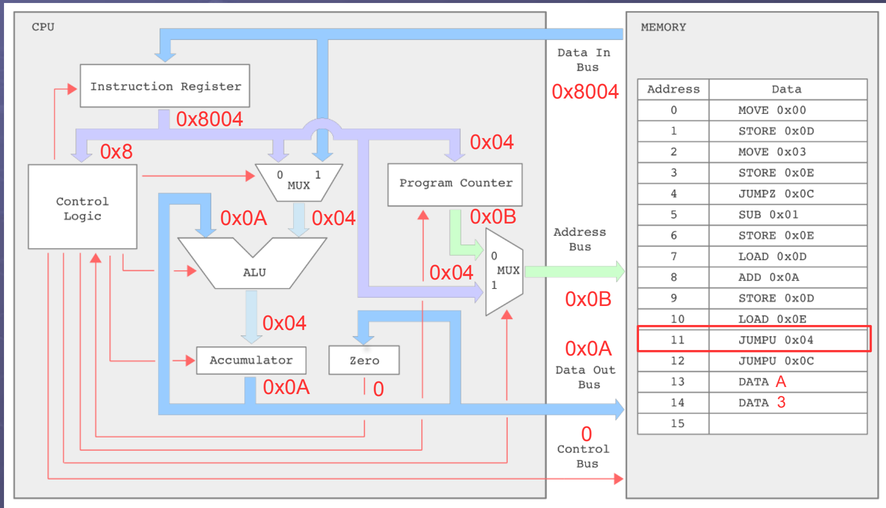
SimpleCPU_v1a : JUMPU



- Fetch : get instruction stored at PC address

Slide 55

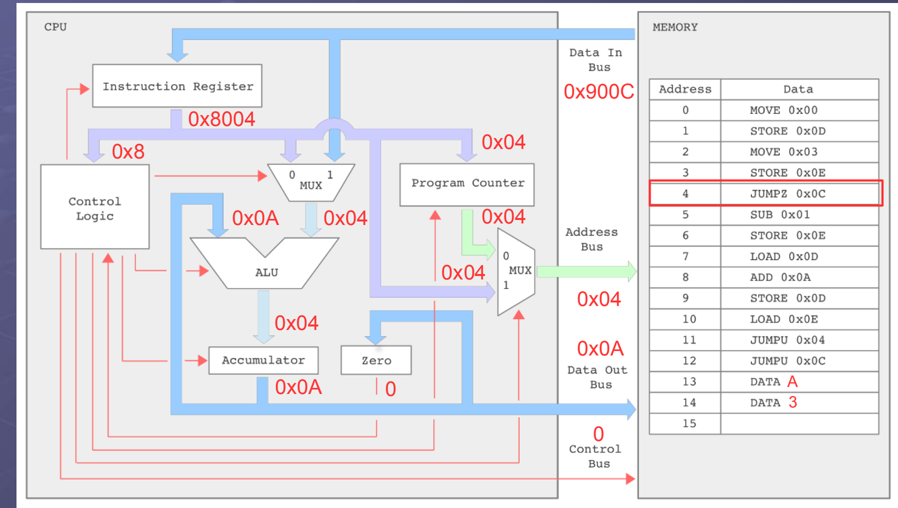
SimpleCPU_v1a : JUMPU



- Decode : PC not incremented, ALU / MUXs not used

Slide 56

SimpleCPU_v1a : JUMPU



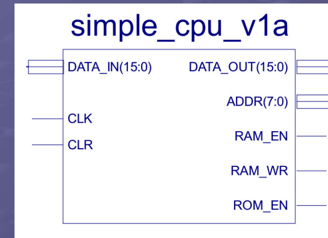
- Execute : PC updated with jump address

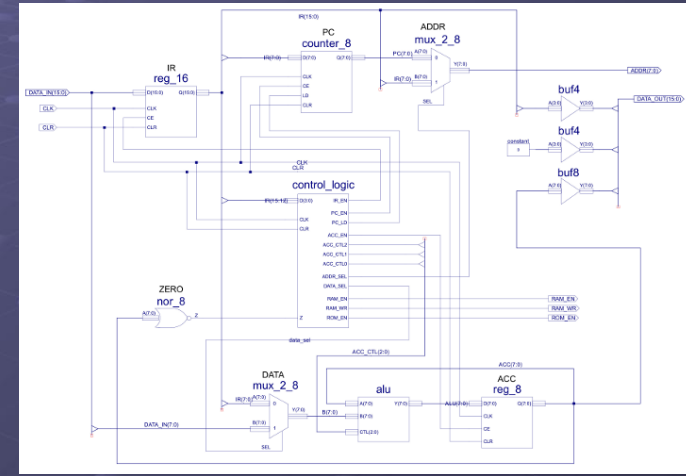
Pause To Consider

- What do instructions do?
  - (A) tells the processor what to do
    - Black box
  - (B) reconfigure hardware to perform a specified task
    - Set up data paths through processor connecting functional units to memory elements
      - **Move** data / variables
    - Configure general purpose processing units (ALU) to perform defined functions
      - **Process** data / variables

- Note: remember the Intel 4004. From one point of view software allows hardware to be reused / reconfigured to emulate more complex functionality e.g. multiplication.

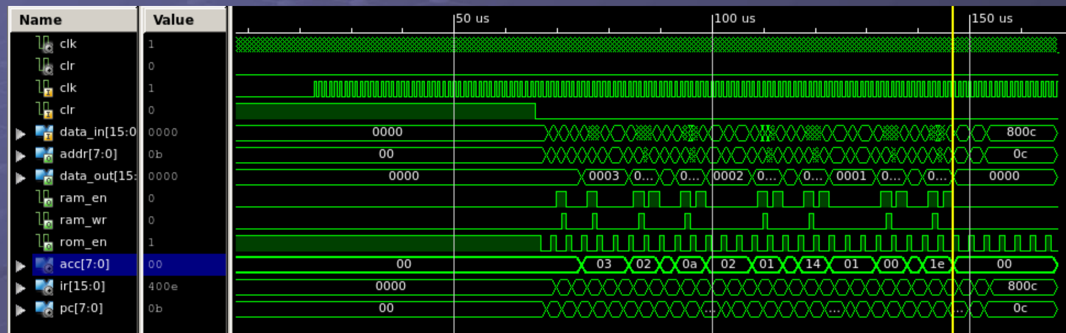
SimpleCPU_v1a





- Xilinx ISE
  - Processor symbol and schematic

Example : SimpleCPU_mul.zip



- Simulation model of the SimpleCPU processor executing the program : 10 x 3 = 30
  - Run-time: approximately 140us at 10MHz

Summary

- Key concepts
  - Fetch – Decode – Execute cycle
    - How an instruction is processed in the computer
    - What an instruction encodes : opcode, operands
    - How data is accessed, addressing modes : immediate, absolute, direct.
  - How an instructions is represented
    - High level language, assembler and machine code
  - How machine level instructions are implemented using a series of micro-instructions.